User's Manual of Flow*

May 29, 2013

1 Overview

The reachability analysis of hybrid systems is a difficult task, since the reachability problem on hybrid systems is not decidable even the variable derivatives are all constant. The tool Flow* computes Taylor Model (TM) flowpipes for a given hybrid system. Each flowpipe is an over-approximation of the reachable set in a small time interval (time step), and the union of them then over-approximates the reachable set in bounded time horizon and jump depth.

Flow* deals with a subclass of general hybrid systems such that (i) the continuous dynamics are given by polynomial Ordinary Differential Equations (ODEs), (ii) a mode invariant or jump guard is defined by a conjunction of polynomial constraints, and (iii) a jump reset is a polynomial mapping.

The high-level structure of Flow* is presented in Figure 1. The tool uses TM based methods including (a) integrating ODEs, (b) computing TM/guard and TM/invariant intersections, (c) aggregating finitely many TMs to compute a set of TM flowpipes which over-approximate the reachable set of the given hybrid system. The tool accepts the following input files:

(i) *a model file* which specifies the reachability task on a continuous or hybrid system;

(ii) *a TM flowpipe file* which contains the state space definition and TM flowpipes.

For a model file, Flow* performs the TM flowpipe construction and generates a plotting file as long as a TM flowpipe file which can be reused by Flow* or other tools. If there is an unsafe set specified, the tool also returns the checking results. For a TM flowpipe file, Flow* calls the TM analyzer module and generates a plotting file. Similarly, a safety checking result may also be returned if an unsafe set is specified.

It is easy to use Flow*. Users may execute

```
$./flowstar < reach.model
```

to let the tool work on a reachability task specified in `reach.model`. If the computation is successful, Flow* terminates with a plotting file (ends with `.plt`) and a TM flowpipe file (ends with `.flow`) which are located in the subdirectory `outputs`. The software Gnuplot is required to visualize a plotting file `flowpipes.plt`. In the tool directory, users may simply execute

```
$gnuplot ./outputs/flowpipes.plt
```

and then an `eps` file will be produced in the subdirectory `images`. Users may change the plot setting or the unsafe set specification in the flowpipe file and

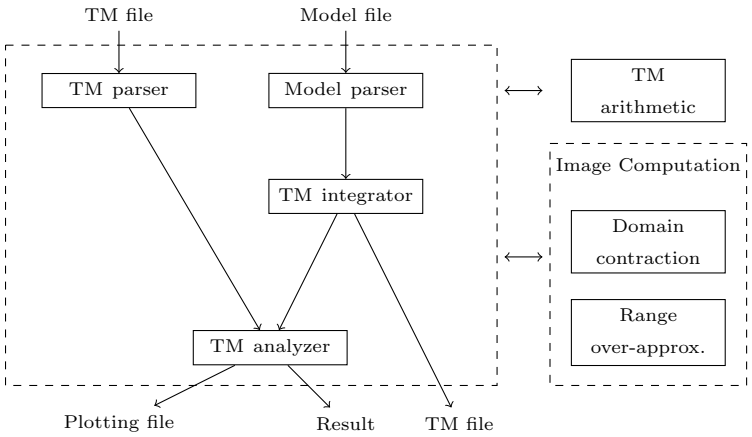


Figure 1: Structure of Flow*

let Flow* do the new analysis job on the original flowpipes:
`$./flowstar < ./outputs/flowpipes.flow`
a new plotting file will be generated in `images`.

Flow* has a simple installation. After the libraries and tools listed in Table 1 properly installed, users only need to type `make` to compile the source code, and an executable file `flowstar` will be generated.

Names	Weblinks
GMP Library	`http://gmplib.org/`
GNU MPFR Library	`http://www.mpfr.org/`
GNU Scientific Library	`http://www.gnu.org/software/gsl/`
GNU Linear Programming Kit	`http://www.gnu.org/software/glpk/`
Bison - GNU parser generator	`http://www.gnu.org/software/bison/`
flex: The Fast Lexical Analyzer	`http://flex.sourceforge.net/`
Gnuplot	`http://www.gnuplot.info/`

Table 1: Libraries and tools needed by Flow*

The technical details of Flow* are described in [1, 2].

2 Model specifications

We introduce the modeling language in Flow* by two examples.

2.1 Modeling language for continuous systems

A Van-der-Pol oscillator can be described by the following 2-dimensional ODE:

$$\begin{cases} \dot{x} &= y \\ \dot{y} &= y - x - x^2 \cdot y \end{cases}$$

We use Flow* to perform a flowpipe construction from the initial set $x \in [1.25, 1.55], y \in [2.25, 2.35]$ with the time horizon $[0, 7]$. The content of the

modeling file is shown as below.

```
continuous system
{
 state var x,y              # declare the state variables

 settings                   # the reachability setting
 {
  fixed steps 0.02          # time step size
  time 7                    # time horizon
  remainder estimation 1e-5 # [-1e-5,1e-5] in each dimension
  QR precondition           # the preconditioning technique
  visualize octagon x,y     # plot the octagon enclosure of the flowpipes
  adaptive orders { min {x:4 , y:4} , max {x:6 , y:6} } # TM orders
  output vanderpol          # name for the output files
  print on                  # display the computation steps
 }

 init                       # specify the initial set
 {
  x in [1.25 , 1.55]
  y in [2.25 , 2.35]
 }

 ode                        # specify the ODE
 {
  x' =  y
  y' =  y - x - x^2*y
 }
}
```

The explanation of the parameters are as follows.

- *Time step size* - the size of TM integration steps. A fixed step size 0.02 can be specified by
  `fixed steps 0.02`
  or a user may want Flow* to perform adaptive step sizes range from 0.01 to 0.05,
  `adaptive steps { min 0.01, max 0.05 }`

- *Time horizon.* A time horizon $[0, T]$ can be specified by
  `time T`

- *Remainder estimation* - the remainder estimation interval used in TM integration. Users may simply write, for example
  `remainder estimation 1e-5`
  to tell Flow* to use $[-10^{-5}, 10^{-5}]$ as the estimation interval in all dimensions. Or more specifically, apply different estimations over the dimensions. For example, the following specification tells Flow* to apply $[-0.002, 0.002]$ to the dimension x and $[-0.1, 0.1]$ to the dimension y.
  `remainder estimation { x:[-2e-3,2e-3], y:[-0.1,0.1] }`
  In adaptive flowpipe construction, the wrapping effect can be controlled by using a set of small estimations.

- *Preconditioning technique.* Flow* provides two preconditioning techniques to control the wrapping effect in TM integration. They can be specified by `QR precondition` and `identity precondition` respectively. For the systems with more than 3 variables, QR preconditioning is *not* suggested to use.

- *Plot setting.* In order to view the reachability results intuitively, FLOW* over-approximates a 2D projection of the flowpipes by two classes of geometric objects: intervals (boxes) and octagons. The image file can be generated by Gnuplot. To view the interval enclosures of the flowpipes in an x-y plane, users may specify
  `visualize interval x,y` To provide a more accurate visualization, FLOW* may generate a grid paving for the flowpipes. The accuracy is determined by the grid number which can be specified by users. For example, users may use the plot setting
  `visualize grid 10 x,y`
  to tell FLOW* to split the domain of a TM flowpipe evenly into 10 parts in each dimension, totally 10^n grids wherein n is the number of the variables, and compute the images of the grids under the flowpipe mapping by interval arithmetic. The three visualizations of the Van-der-Pol oscillator example is presented in Figure 2.

- *TM orders* - the orders of the TM arithmetic. Theoretically, FLOW* may always achieve a better accuracy with higher TM orders as long as proper remainder estimations. A fixed order for all state variables can be specified like
  `fixed orders 5`
  or the user may like to try uniform adaptive orders
  `adaptive orders { min 3 , max 6 }`
  It is quite common that a system may have several variables change at very different rates. In that case, FLOW* allows users to specify different orders or order ranges over the state variables. For example,
  `fixed orders { x:3 , y:4 }`
  and
  `adaptive orders { min { x:2 , y:3 } , max{ x:5 , y:4 } }`

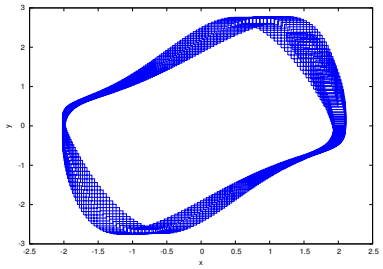
- *Output specification.* After a successful computation, FLOW* generates two files `name.plt` and `name.flow` in the output directory such that `name` can be specified in the model file by
  `output name`

- *Print out.* The display of the computation steps can be turned on or off by specifying `print on` or `print off` respectively.

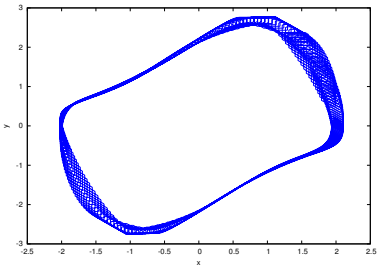
With proper adaptive parameters, users may obtain a good efficiency with very little loss of accuracy.

The initial set of a continuous system can be an interval or a TM. An interval initial set is given by specifying each variable an interval, for example
`x in [1,2]  y in [-1,1]`
The definition of a TM initial set consists of three parts: (1) TM variable declaration, (2) the TM polynomial and remainder (3) the TM domain. An example of 2 state variables `x,y` is given as below.

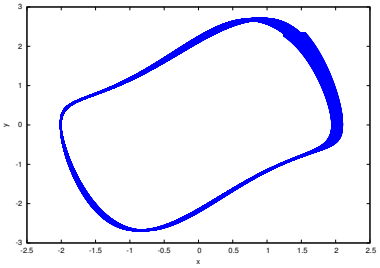
```
tm var x0 , x1 , x2
x = 1 + x1^2 - x2 + [0,0]
y = x0^3 - x1 + [0,0]
x0 in [-0.2,0.2]
x1 in [-0.2,0.2]
x2 in [-0.1,0.1]
```



(a) intervals



(b) octagons



(c) grids

Figure 2: Van-der-Pol oscillator

Currently, Flow* only accepts polynomial ODEs. For a state variable `x`, its ODE can be specified by `x' = p` such that `p` is a polynomial over the state variables. The syntax of polynomials in Flow* is given as below.

```
p := p+p | p-p | p*p | -p | (p) | p^n | x | r
```

such that `n` is a positive integer, `x` is a variable and `r` is a real number. The priorities over the operators are shown in Table 2.

In addition, an unsafe set can be optionally specified after the reachability definition:

```
unsafe set
{
 # polynomial constraints
 p1 <= c1
 p2 >= c2
 p3 in [c3,c4]
 ...
}
```

^	4
- (prefix)	3
*	2
+	1
- (infix)	1

Table 2: Priorities of the operators

The set is defined by the conjunction of the user given polynomial constraints, each of which is of the form `p` $\sim$ `c` or `p in [c1,c2]` wherein `p` is a polynomial over the state variables, `c,c1,c2` are constants and $\sim \in \{<=, >=, =\}$. After performing the reachability computation, Flow* checks the intersection of the flowpipes and the unsafe set. If no intersection is detected, then the tool reports `SAFE`, otherwise it reports `UNKNOWN`.

2.2 Modeling language for hybrid systems

In this section, we present the model file of a collision avoidance maneuver of two aircraft at a fixed altitude [3]. In the beginning, both of the aircraft are in straight flight with a relative heading. When the aircraft come too close, that is the distance of them is blow a specified value, the controller will make an instantaneous heading change of 90° on both of them, and then the two aircraft will complete a π-time semicircular arc flying. Afterwards, both aircraft make another 90° instantaneous heading change and resume their headings in the very beginning. Such a system can be modeled by a 3-mode hybrid system, see Figure 3.

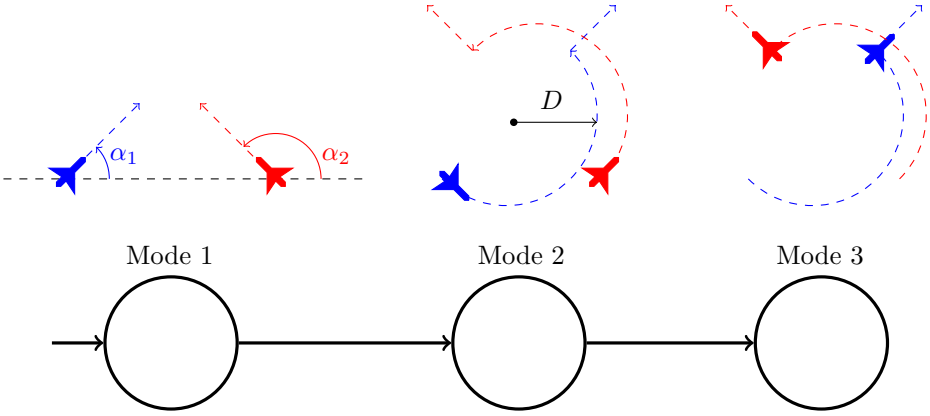


Figure 3: Hybrid automaton of the collision avoidance maneuver

The state variables are as follows:

x_1, y_1 :	the coordinates of the first aircraft
α_1 :	the heading angle of the first aircraft
x_2, y_2 :	the coordinates of the second aircraft
α_2 :	the heading angle of the second aircraft
z :	timer for the semicircular arc flying

Here, a heading angle is the angle between the heading direction and the vector $(1, 0)^T$ in the coordinate map.

The dynamics of x_1, y_1, x_2, y_2 in each mode are

$$\dot{x}_1 = v_1 \cdot \cos\alpha_1,\ \dot{y}_1 = v_1 \cdot \sin\alpha_1,\ \dot{x}_2 = v_2 \cdot \cos\alpha_2,\ \dot{y}_2 = v_2 \cdot \sin\alpha_2$$

The angles α_1, α_2 only changes at rate 1 in Mode 2, and so is the timer z. To make Flow* work on the example, we need to equivalently reformalize the dynamics by polynomials. To do so, four new variables c_1, s_1, c_2, s_2 are introduced to substitute the trigonometric terms $\cos\alpha_1, \sin\alpha_1, \cos\alpha_2, \sin\alpha_2$ respectively. Then, the dynamics in Mode 1 and 3 becomes

$$\begin{array}{lllll} \dot{x}_1 = v_1 \cdot c_1, & \dot{y}_1 = v_1 \cdot s_1, & \dot{c}_1 = 0, & \dot{s}_1 = 0, & \dot{\alpha}_1 = 0, \quad \dot{z} = 0, \\ \dot{x}_2 = v_2 \cdot c_2, & \dot{y}_2 = v_2 \cdot s_2, & \dot{c}_2 = 0, & \dot{s}_2 = 0, & \dot{\alpha}_2 = 0 \end{array}$$

the dynamics in Mode 2 is

$$\begin{array}{lllll} \dot{x}_1 = v_1 \cdot c_1, & \dot{y}_1 = v_1 \cdot s_1, & \dot{c}_1 = -s_1, & \dot{s}_1 = c_1, & \dot{\alpha}_1 = 1, \quad \dot{z} = 1, \\ \dot{x}_2 = v_2 \cdot c_2, & \dot{y}_2 = v_2 \cdot s_2, & \dot{c}_2 = -s_2, & \dot{s}_2 = c_2, & \dot{\alpha}_2 = 1 \end{array}$$

For the jump from Mode 1 to Mode 2, the guard is $(x_1-x_2)^2+(y_1-y_2)^2 \leq D^2$, and the reset mapping is

$$\alpha_1' := \alpha_1 - \frac{\pi}{2},\ \alpha_2' := \alpha_2 - \frac{\pi}{2},\ c_1' := s_1,\ s_1' := -c_1,\ c_2' := s_2,\ s_2' := -c_2$$

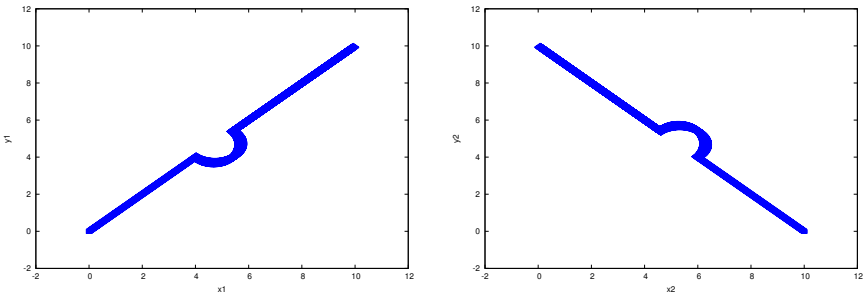
For the jump from Mode 2 to Mode 3, the guard is $z = \pi$, and the reset mapping is same as the above one. Our model file is shown as below wherein we choose $v_1 = v_2 = 1$ and $D = 2$.

```
hybrid system
{
 state var x1, y1, c1, s1, a1, x2, y2, c2, s2, a2, z

 settings
 {
  fixed steps 0.02
  time 15
  remainder estimation 1e-6
  identity precondition
  visualize interval x1,y1
  adaptive orders { min 2 , max 5 }
  output aircraft
  max jumps 2    # the bound on the jump depth
  print on
 }

 modes
 {
  mode1
  {
   ode
   {
    x1' = c1  y1' = s1  c1' = 0   s1' = 0  a1' = 0  x2' = c2
    y2' = s2  c2' = 0   s2' = 0   a2' = 0  z' = 0
   }
   inv
   {
    (x1 - x2)^2 + (y1 - y2)^2 >= 4
   }
  }
  mode2
  {
   ode
   {
    x1' = c1  y1' = s1  c1' = -s1 s1' = c1 a1' = 1  x2' = c2
    y2' = s2  c2' = -s2 s2' = c2  a2' = 1  z' = 1
   }
   inv
   {
    z <= 3.1415
   }
  }
  mode3
  {
   ode
   {
    x1' = c1  y1' = s1  c1' = 0   s1' = 0  a1' = 0  x2' = c2
    y2' = s2  c2' = 0   s2' = 0   a2' = 0  z' = 0
   }
   inv    # no invariant
   {
   }
```

```
  }
 }

 jumps
 {
  start mode1
  end mode2
  guard { (x1 - x2)^2 + (y1 - y2)^2 = 4 }
  reset { a1' := a1 - 1.57075  a2' := a2 - 1.57075  s1' := -c1
          c1' := s1            s2' := -c2           c2' := s2 }
  parallelotope aggregation { [x1:1 , y1:1] [x2:1 , y2:1] }

  start mode2
  end mode3
  guard { z = 3.1415 }
  reset { a1' := a1 - 1.57075  a2' := a2 - 1.57075  s1' := -c1
          c1' := s1            s2' := -c2           c2' := s2 }
  parallelotope aggregation {  }  # the template is auto selected
 }
 init
 {
  mode1 # initial mode
  {
   x1 in [-0.1,0.1]          y1 in [-0.1,0.1]           c1 in [0.70710,0.70711]
   s1 in [0.70710,0.70711]   a1 in [0.785374,0.785376]  x2 in [9.9,10.1]
   y2 in [-0.1,0.1]          c2 in [-0.70711,-0.70710]  s2 in [0.70710,0.70711]
   a2 in [2.35612,2.35613]    z in [0,0]
  }
 }
}

unsafe set  # there is a collision in a mode
{
 mode1
 {
  x1 - x2 = 0
  y1 - y2 = 0
 }
 mode2
 {
  x1 - x2 = 0
  y1 - y2 = 0
 }
 mode3
 {
  x1 - x2 = 0
  y1 - y2 = 0
 }
}
```

The starting position of the first aircraft is the region $[-0.1, 0.1] \times [-0.1, 0.1]$ in the coordinate map. For the second aircraft, the initial position is the region $[9.9, 10.1] \times [-0.1, 0.1]$. The initial heading angle of the first aircraft is $\frac{\pi}{4}$ and that of the second aircraft is $\frac{3}{4}\pi$. There are at most two jumps in the hybrid automaton, so we bound the jump depth by 2: `max jumps 2`. We set the time horizon $[0, 15]$ such that the system will definitely in mode 3 after 15 time units. The octagon enclosures of the two aircraft trajectories are presented in Figure 4, no collision is detected by FLOW*.

In a hybrid model file, the discrete modes and jumps are declared in `modes { ... }` and `jumps { ... }` respectively. The specification of a mode is of



(a) First aircraft (b) Second aircraft

Figure 4: Collision avoidance maneuver

the following structure:

```
name # user-defined name of the mode
{
 ode
 {
  # polynomial ODE of the mode
 }
 inv
 {
  # polynomial constraints
 }
}
```

A jump from mode `m1` to mode `m2` is defined by

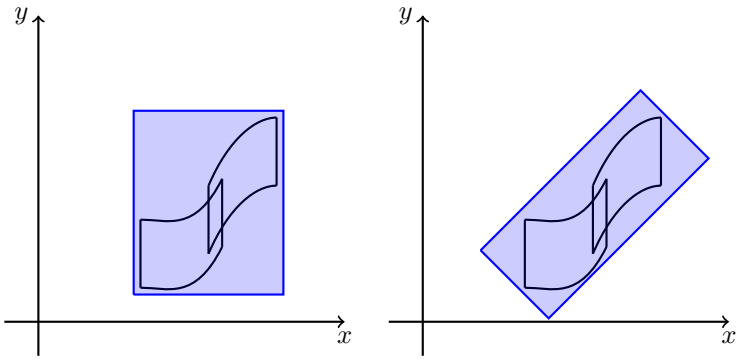
```
start m1
end m2
guard
{
 # polynomial constraints
}
reset
{
 # polynomial reset mapping with uncertainties
}
... # aggregation setting
```

FLOW* accepts polynomial reset mappings of the form `x' := p` wherein `x` is a state variable and `p` is a polynomial over the state variables. Additionally, users may also include an interval uncertainty, that is `x' := p + [a,b]`. An unspecified variable will be assigned an identity mapping, that is `x' := x`.

To relieve the computation burden, it is always necessary to aggregate the flowpipes after a jump. Since TMs are not closed under union, aggregating a set of TMs is not an easy job. FLOW* provides the following schemes to compute a parallelotopic aggregation which can be efficiently transformed into an order 1 TM with zero remainder interval.

The key part of the parallelotopic aggregation is the template selection. For a d-dimensional parallelotope, its orientation can be determined by d linearly independent vectors, the set of which is called *template*. We give an example in Figure 5. The two flowpipes can be over-approximated by a parallelotope with the template $\{(1,0)^T,(0,1)^T\}$, or the template $\{(1,1)^T,(1,-1)^T\}$. It is obvious

that the aggregation accuracy plays an important role in the subsequent flowpipe construction work. FLOW* provides the following options to select a template for a jump.



(a) template $\{(1,0)^T, (0,1)^T\}$ (b) template $\{(1,1)^T, (1,-1)^T\}$

Figure 5: Example of flowpipe aggregation

- *Interval aggregation.* FLOW* computes a box enclosure for the flowpipe union, similar to the example shown in Figure 5(a). The specification is `interval aggregation`
- *General parallelotopic aggregation.* Users may simply specify `parallelotope aggregation { }` then FLOW* will determine the template based on the mode invariant, jump guard, flow direction and so on. Users are also allowed to provide some preferred vectors, for a 3-dimensional example `parallelotope aggregation { [x:1,y:1] [z:2] }` wherein `[x:1,y:1]` denotes the vector $(1,1,0)^T$ in the state space, and `[z:2]` denotes $(0,0,2)^T$. The unspecified state variables will be assigned zero, so users may not need to specify the zero components. Since only the directions of vectors are concerned, FLOW* will normalize the user-given vectors and consider them on the highest priority in determining the templates.

To the best of our knowledge, there is no best approach to choose templates. Then more heuristics will be added in the future.

More benchmarks are described on the homepage of FLOW*, please visit

`http://ccis.colorado.edu/research/cyberphysical/taylormodels/`

References

[1] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proc. of the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*, pages 183–192. IEEE, 2012.

[2] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of the 25th International Conference on Computer Aided Verification (CAV'13)*, LNCS. Springer, 2013.

[3] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Proc. of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 310–323. Springer, 2000.